Fall 9-1-1984

On Finite Automata with a Unison Working Tape ; CU-CS-278-84

Andrzej Ehrenfeucht
University of Colorado Boulder

Grzegorz Rozenberg
University of Leiden

ON FINITE AUTOMATA WITH A UNISON WORKING TAPE

by

A. Ehrenfeucht* and G. Rozenberg**

CU-CS-278-84 September, 1984

All correspondence to the second author.

*Department of Computer Science, University of Colorado Boulder, CO 80309

**Institute of Applied Mathematics and Computer Science, University of Leiden, Leiden, The Netherlands

ABSTRACT.

In this paper we consider the machine model which consists of a 2-way finite automaton equipped with a working tape. This working tape works in *unison* with the input tape of the 2-way finite automaton in the sense that the moves of heads of both tapes are always of the same type: either both heads remain at the same place or both heads move to the left or both heads move to the right. As usual the input tape head is only reading while the working tape head is a read-write head.

The main result of the paper states that adding to a 2-way finite automaton a unison push-down tape does not extend the recognition power - i.e., such an automaton recognizes regular languages only. As a corollary of this result one gets directly: (1) the Shepherdson result that two-way finite automata recognize only regular languages and (2) the Büchi result (even its extended Greibach-Kratko version) that regular canonical systems generate only regular languages.

We also discuss a couple of extensions of the "unison push-down" model and in particular we show that even equipping a 2-way finite automaton with a unison stack tape (i.e. the nondestructive reading within the push-down tape is allowed) still leaves are with regular languages only.

INTRODUCTION.

A finite automaton (fa) constitutes the most fundamental machine model in the theory of computation. By removing various constraints on the way finite automaton operates and/or adding to it various types of storage one gets many other (if not all) models of (sequential) computation (see, e.g., [6] and [12]).

One of the earliest studied extensions of the fa model is a 2-way finite automaton where the reading moves on the input tape can be both forward (to the right) and backward (to the left). It was proved by M. Rabin and J. Shepherdson ([13], see also [9]) that 2-way finite automata are not more powerful than ordinary (1-way) finite automata. On the other hand various other models were studied where a fa is equipped with an additional storage tape(s) like e.g. counter, push-down, stack, etc. Most (if not all) of these extensions are increasing the accepting power of the basic fa model - languages ("far") beyond regular can be generated in this way. Also, analogous machine models where these types of the additional storage are added to a 2-way fa were studied in the literature: 2-way push-down automata or 2-way stack automata are examples of such machines (see, e.g., [6]). In most cases adding a given type of an additional storage to a 2-way fa turns out to be more powerful than adding the same type of storage to an ordinary (1-way) fa.

Often adding a number of different types of storage (tapes) to the fa model leads to a too powerful model: one can accept all recursively enumerable languages. Therefore various restricted ways of using auxiliary storages were studied in the literature: e.g., the heads scanning two different types of storage tapes were moving synchronously (in unison) and/or at the beginning of a computation one of the storage tapes would be "preset" by a specific content - and so it would become a "checking tape" (see, e.g., [2], [3], [5]).

In this paper we continue this "constraining-the-use-of-a-storage-tape" line of reasoning but now we require that the *input tape and the storage (working) tape move in unison.* We impose this restriction on the 2-way fa model using various types of working tape.

The central result of this paper (proved in Section 3) states that 2-way finite automata equipped with a unison push-down tape accept regular languages only. This result clearly generalizes the Rabin-Shepherdson theorem and moreover we show how the result by Büchi ([1], see also [4], [7] and [11]) that regular canonical systems generate regular languages only can be easily obtained from our theorem (see Section 4). In Section 5 we discuss other unison extensions of the 2-way fa model and among others we show that adding a unison stack to a 2-way fa still leaves one with regular languages only.

PRELIMINARIES.

We assume the reader to be familiar with basic automata and formal language theory (see e.g. [6] or [11]). We use mostly standard notation. Perhaps only the following notational matters should be pointed out.

N denotes the set of natural numbers. For a set A, 2^A denotes the family of its subsets and $\#A$ denotes the cardinality of A. To make our notation simpler we will often notationally identify a singleton set with its element; i.e., if $A = \{a\}$ then we may write a rather than $\{a\}$.

For a word x, $|x|$ denotes its length. If $x \in \Sigma^+$ for an alphabet Σ, $x = a_1 a_1 \cdots a_n$ for some $n \geq 1$ and $a_i \in \Sigma$ for $0 \leq i \leq n$, then $i(n) = a_i$ for $0 \leq i \leq n$. Λ denotes the empty word. For a letter a, a^ω denotes the 1-way (to the right) infinite word consisting of a's only.

For an equivalence relation E, $ind(E)$ denotes its index. Given an alphabet Σ and a language $K \subseteq \Sigma^*$, E_K denotes the Nerode equivalence relation (see, e.g., [6]) defined by: for $x_1 y \in \Sigma^*$, $xE_K y$ if and only if for each $u \in \Sigma^*$ ($xu \in K$ if and only if $yu \in K$).

The following result by A. Nerode (see, e.g., [6]) provides a fundamental characterization of regular languages.

Proposition 1. A language K is regular if and only if E_K is of finite index. ▪

We will use $\mathbf{L}(RE)$, $\mathbf{L}(CS)$ and $\mathbf{L}(REG)$ to denote the classes of recursively enumerable, context sensitive and regular languages, respectively.

2. General Model.

Our general model may be depicted as follows.

Figure 1.

At each point of a computation the common head sees the same position at both tapes: the input tape and the working tape both of which are 1-way infinite tapes - see Figure 1. Depending on the current state (q) the automaton decides to change its state to q' (may be equal to q), to change the currently observed symbol on the working tape to b_i' (may be equal to b_i) and to change the position of the common head either to the left (to the position $(i-1)$) or to the right (to the position $(i+1)$) or to stay stationary (i.e., to remain at the i'th position). Thus the "input part of the common head" is read - only while the "output part of the common head" is read-and-write. The automaton may never go to the left beyond the left-end-marker $\$$. In the "initial situation":

Figure 2.

the automaton is in its initial state (q_{in}) and reads the first cells of the tapes (that is the cells immediately to the right of the left-end-marker cells containing $\$$) - for some $r \geq 0$ the first r cells of the input tape contain symbols from the input alphabet (hence the word $a_1 \cdots a_r$) and all other cells ($n \geq r$) are blank (i.e., they contain the blank symbol), while all the cells on the working tape (to the right of the $\$$ cell) contain blank symbols only. The word $a_1 \cdots a_r$ is accepted by the automaton if in a finite sequence of moves the automaton enters the 0'th cells (i.e., the $\$$ cells in one of its accepting states):

Figure 3.

The language of the automaton consists of all words accepted by it.

Formally this model is defined as follows.

Definition. A 2-way automaton with a unison working tape, a 2faut for short, is a system

$\mathbf{A} = (\Delta, \Sigma, Q, F, q_{in}, \delta)$, where

Δ is a finite nonempty alphabet, called the *input alphabet (of* **A***)*,

Σ is a finite nonempty alphabet, called the *working alphabet (of* **A***)*,

Q is a finite nonempty set, called the *set of states (of* **A***)*,

$F \subseteq Q$ is called the set of *accepting states (of* **A***)*,

$q_{in} \in Q$ is called the *initial state (of* **A***)*,

δ is a (partial) function from $(\Delta \cup \{B, \$\}) \times (\Sigma \cup \{B, \$\}) \times Q$ into $2^{(\Sigma \cup \{\$, B\}) \times \{-1, 0, +1\} \times Q}$

satisfying the condition: if $(y', n, q') \in \delta(x, y, q)$ then

(i) $y' = \$$ if and only if $x = \$$ if and only if $y = \$$, and

(ii) $x = \$$ implies $n = +1$.

δ is called the *transition function (of* **A***)*. ▪

Remark.

(1) Symbols B and $\$$ are special reserved symbols in the sense that they will be used in the same way in all faut's we will consider (and so they will never be included in either the input or the working alphabet of a utfa). The symbol B stands for "blank" and the symbol $\$$ for the "left end marker".

(2) Our conditions on δ make it sure that $\$$ plays only the role of the left-end-marker. We have found it convenient to require that if a move on $\$$ is needed, then it is the "bounce-to-the-right" move; in general one could allow stationary moves on $\$$ (i.e., in (2) above $n = 0$ would also be allowed).

(3) The elements of $(\Delta\cup\{B, \$\})\times(\Sigma\cup\{B, \$\})\times Q$ are called *situations (of* **A***)* and the elements of $(\Sigma\times\{B, \$\})\times\{-1, 0, +1\}\times Q$ are called *actions (of* **A***)*; hence δ is a function from situations into sets of actions. ▪

Definition. Let $\mathbf{A} = (\Delta, \Sigma, Q, F, q_{in}, \delta)$ be a 2faut.

(1) A *configuration (of* **A***)* is a four-tuple $\tau = (\alpha, \beta, m, q)$, where $\alpha \in \$\Delta^* B^\omega$, $\beta \in \$\Sigma^* B^\omega$, $m \geq 0$ and $q \in Q$; τ is called *initial* if $q = q_{in}$, $m = 1$ and $\beta = \$B^\omega$ and τ is called *accepting* if either $\alpha \in \$\Delta^+ B^\omega$, $q \in F$ and $m = 0$ or $\alpha = \$B^\omega$, $q \in F$ and $m = 1$. We use *input* (τ) to denote $w \in \Delta^*$ such that $\alpha = \$wB^\omega$, *work* (τ) to denote $z \in \Sigma^*$ such that $\beta = \$zB^\omega$, *pos* (τ) to denote m and *st* (τ) to denote q.

(2) Let $\tau = (\alpha, \beta, m, q)$, $\tau' = (\alpha', \beta', m', q')$ be configurations. We say that τ *directly yields* τ' *(in* **A***)*, written $\tau \vdash_{\mathbf{A}} \tau'$, if $\alpha' = \alpha$ and there exists an action $(y_1, n_1, q_1) \in \delta(m(\alpha), m(\beta), q)$ such that

(i) $m' = m + n_1$,

(ii) $q' = q_1$ and

(iii) $m(\beta') = y_1$ and, for all $n \in \mathbf{N}$, $n(\beta') = n(\beta)$ whenever $n \neq m$

(We also say that the *action* (y_1, n_1, q_1) *is used in directly yielding* τ' *from* τ.)

(3) A *computation (in* **A***)* is a sequence $\mu = \tau_0, \tau_1, \ldots, \tau_l$, $l \geq 1$, of configurations such that

(i) either $l = 0$ and *input* (τ_0) = *work* $(\tau_0) = \Lambda$ or $l \geq 1$ and *input* $(\tau_0) \neq \Lambda$, and

(ii) if $l \geq 1$, then $\tau_i \vdash_{\mathbf{A}} \tau_{i+1}$ for all $0 \leq i \leq l-1$.

$l+1$ is referred to as the *length of* μ and denoted by $lg(\mu)$.

We say then that τ_0 *yields* τ_l *(in* **A***)* and write $\tau_0 \vdash_{\mathbf{A}}^{+} \tau_l$.

Each pair (τ_i, τ_{i+1}), $0 \leq i \leq l-1$ is called a *step of* μ (or more precisely the i'th *step of* μ) and if the action A is used in directly yielding τ' from τ then we say that A is *used in the i'th step of* μ. The state $st(\tau_l)$ is called the *last state of* μ and denoted by $lst(\mu)$.

If τ_0 is initial, $input(\tau_0) = w$ and τ_l is accepting then we say that μ is a *w-accepting computation*.

(4) The *language of* **A**, denoted $L(\mathbf{A})$, is the set

$\{ w \in \Delta^* :$ there exists a w-accepting computation in **A** $\}$. ■

Remark. Note that the Λ-accepting computation is of length 0 and then $q_{in} \in F$. ■

The class of languages of all 2 faut's is denoted by $\mathbf{L}(2FAUT)$.

Since it is obvious that each Turing machine can be simulated by a 2faut, we have the following result.

Theorem 1. $\mathbf{L}(2FAUT) = \mathbf{L}(RE)$. ■

3. PUSH-DOWN WORKING TAPE.

In this section we consider a special case of the 2faut model, where the working tape is a push-down tape. We demonstrate that such automata define regular language, only.

Definition. A 2faut $\mathbf{A} = (\Delta, \Sigma, Q, F, q_{in}, \delta)$ is called *2-way finite automaton with a unison push-down tape*, a 2faupd for short, if, for each situation (x, y, q) and each action (y', n, q'), $(y', n, q') \in \delta(x, y, q)$ implies that

(1) $x \neq \$$ and

(2) $y' = B$ if and only if $n = -1$. ■

The class of languages of all 2faupd's is denoted by $\mathbf{L}(2FAUPD)$.

We prove now the central theorem of this paper.

Theorem 2. $\mathbf{L}(2FAUPD) = \mathbf{L}(REG)$.

Proof.

Clearly $\mathbf{L}(REG) \subseteq \mathbf{L}(2FAUPD)$.

Hence to prove the theorem it suffices to show that $\mathbf{L}(2FAUPD) \subseteq \mathbf{L}(REG)$. This will be accomplished by demonstrating that for each 2faupd $\mathbf{A}$ the Nerode relation $E_{L(\mathbf{A})}$ is of finite index - the result follows then from Proposition 1.

Let $\mathbf{A} = (\Delta, \Sigma, Q, F, q_{in}, \delta)$ be a 2faupd.

First we need a number of auxiliary notions.

For $q \in Q$, $\mathbf{A}(q)$ is the 2faupd $(\Delta, \Sigma, Q, Q, q, \delta)$.

For $C \subseteq Q \times Q$, $\mathbf{A}_C$ is the 2faupd $(\Delta, \Sigma, Q, F, q_{in}, \delta_C)$, where δ_C is defined as follows:

for a situation (x, y, q)

$$\delta_C(x, y, q) = \begin{cases} (y', n, q') & \text{if } x \neq B \text{ and } (y', n, q') \in \delta(x,y,q), \\ (B, -1, q') & \text{if } x = B, y = B \text{ and } (q, q') \in C, \\ \text{undefined} & \text{otherwise.} \end{cases}$$

For $w \in \Delta^+$, $R(w) = \{C \subseteq Q \times Q : w \in L(\mathbf{A}_C)\}$.

For $w \in \Delta^*$,

$C_w = \{(q, q') \in Q \times Q$: there exists a w-accepting computation μ in $\mathbf{A}(q)$ such that $lst(\mu) = q'\}$.

Now we prove two lemmas.

Lemma 1. For each $w \in \Delta^+$ and each $u \in \Delta^*$, $wu \in L(\mathbf{A})$ if and only if $C_u \in R(w)$.

Proof of Lemma 1.

Let $w \in \Delta^+$ and let $u \in \Delta^+$.

(i) Assume that $wu \in L(G)$ and consider a wu-accepting computation $\mu = \tau_0, \tau_1, \ldots, \tau_m$ in $\mathbf{A}$.

We note that if $i, j \in \{0, \ldots, m\}$ are such that $1 < i, j < m$ $pos(\tau_i) = |w|+1$, $st(\tau_i) = q$, $pos(\tau_j) = |w|$, $st(\tau_j) = q'$ and $pos(\tau_k) \geq |w|+1$ for all $i \leq k < j$, then $(q, q') \in C_u$. Consequently $w \in L(\mathbf{A}_{C_u})$ and so $C_u \in R(w)$.

(ii) Assume that $C_u \in R(w)$, hence $w \in L(\mathbf{A}_{C_u})$.

Let $\mu = \tau_0, \ldots, \tau_l$ be a w-accepting computation in $\mathbf{A}_{C_u}$.

If μ is such that, for all $0 \leq j \leq m$, $pos(\tau_j) \leq |w|$, then μ is actually a wu-accepting computation in $\mathbf{A}$; consequently $wu \in L(\mathbf{A})$.

Otherwise we construct a wu-accepting computation ζ in $\mathbf{A}$ as follows.

The first element ρ_0 of ζ is $(\$wuB^\omega, \$B^\omega, 1, q_{in})$. Let $i_1, j_1 \in \{1, \ldots, m-1\}$ be the smallest indices such that $pos(\tau_{i_1}) = |w|+1$, $j_1 > i_1$, $pos(\tau_{j_1}) = |w|$ and for all $i_1 \leq k \leq j_1$ $pos(\tau_k) \geq |w|+1$ (we call $\tau_0, \ldots, \tau_{i_1-1}$ the *first inside block of* μ and $\tau_{i_1}, \ldots, \tau_{j_1-1}$ the *first outside block of* μ).

Now we choose a u-accepting computation κ_1 in $G(st(\tau_{i_1}))$ such that $lst(\kappa_1) = st(\tau_{j_1})$ and construct the initial part of ζ as follows. The first (i_1+1)

steps of ζ are obtained using the sequence of actions from the first i_1 steps of μ (hence the sequence of actions from the first inside block) and then the next $(lg(\kappa_1)-1)$ steps of ζ are obtained by using the sequence of actions from the steps of κ_1.

In this way we have "converted" the initial portion $\tau_0, \ldots, \tau_{j_1-1}$ of μ to an initial portion $\rho_0, \ldots, \rho_{i_1+lg(\kappa_1)-1}$ of ζ.

We iterate the above procedure in such a way that in the remaining portion of μ we distinguish again its first inside and its first outside block (hence the "second inside block of μ" and the "second outside block of μ") and convert them into portions of ζ as above. This iteration is repeated as long as possible.

Finally let r be the largest index such that $pos(\tau_{r-1}) = |w|+1$, $pos(\tau_r) = |w|$ and for all $r \leq k \leq m$, $pos(\tau_k) \leq |w|$. The sequence of actions from the last $(lg(\mu)-r)$ steps of ζ is the sequence of actions form the last $(lg(\mu)-r)$ steps of μ. In this way we have obtained a computation ζ in **A** which is clearly a wu-accepting computation in **A**; consequently $wu \in L(\mathbf{A})$.

Thus assuming that $C_u \in R(w)$ we get $wu \in L(\mathbf{A})$.

The lemma follows now from (i) and (ii). ▪

Lemma 2. For each $w_1, w_2 \in \Delta^+$, if $R(w_1) = R(w_2)$, then $w_1 E_{L(\mathbf{A})} w_2$.

Proof of Lemma 2.

Let $w_1, w_2 \in \Delta^+$.

Assume that $R(w_1) = R(w_2)$.

Then, for each $u \in \Delta^*$, $C_u \in R(w_1)$ if and only if $C_u \in R(w_2)$,

and so by Lemma 1

for each $u \in \Delta^*$, $w_1 u \in L(\mathbf{A})$ if and only if $w_2 u \in L(\mathbf{A})$

and consequently $w_1 E_{L(\mathbf{A})} w_2$. ▪

Now we proceed with the proof of Theorem 2 as follows.

First of all we notice that for each $w \in \Delta^+$, $\#R(w) \leq 2^{e^2}$, where $e = \#Q$. Consequently, by Lemma 2 (and taking into account the empty word - note that in Lemma 2 only nonempty words are considered) we get $ind(E_{L(\mathbf{A})}) \leq 2^{2^{e^2}}+1$. Now the theorem follows from Proposition 1. ∎

4. COROLLARIES.

In this section we demonstrate that two quite basic results from automata theory can be obtained as direct corollaries of Theorem 2.

The first of these results is the classic result by Rabin and Shepherdson ([13], see also [9]) concerning the power of two-way automata.

Corollary 1. The class of languages accepted by 2-way finite automata equals the class of regular languages.

Proof.

This follows directly from Theorem 2 and the observation that a 2-way finite automaton is essentially a 2faupd $\mathbf{A} = (\Delta, \Sigma, Q, F, q_{in}, \delta)$ such that $\#\Sigma = 1$ and, for each situation (x, y, q) and each action (y', n, q'), if $(y', n, q') \in \delta(x, y, q)$, then $x = B$ implies $n = -1$. (Also, in the original definition of a 2-way finite automaton it is required that the automaton "goes off" the right end of the input tape - however it is easily seen that this difference is insignificant). ■

Our second corollary concerns regular canonical systems first considered by E. Post ([8]) and then investigated in depth by R. Büchi ([1]). Regular canonical systems are quite fundamental in automata theory (see, e.g., [5] and [11]) and recently they were shown to be very basic in building up a unified theory of grammars and automata ([10]).

First we recall the definition of a regular canonical system - our formulation follows the one from [5] (see also [11]), which is somewhat more general than the Buchi formulation ([1]).

Definition.

(1) A *(left-)regular canonical system* is a quintuple $G = (\Omega, \Phi, U, V, P)$, where Ω is a finite nonempty alphabet, $\Phi \subseteq \Omega$, $U, V \subseteq \Omega^*$ and P is a finite set of produc-

tions of the form (w, z) where $w, z \in \Omega^*$.

(2) For words $\alpha, \beta \in \Omega^*$, α *directly derives* β in G, written $\alpha \underset{G}{\Longrightarrow} \beta$, if there exists a production (w, z) in P such that $\alpha = \alpha' w$ and $\beta = \alpha' z$; we say that α *derives* β in G, written $\alpha \overset{*}{\underset{G}{\Longrightarrow}} \beta$, if either $\alpha = \beta$ or $\alpha \overset{+}{\underset{G}{\Longrightarrow}} \beta$ where $\overset{+}{\underset{G}{\Longrightarrow}}$ is the transitive closure of $\underset{G}{\Longrightarrow}$.

(3) The *language of* G is the set

$$L(G) = \{x \in \Phi^* : u \overset{*}{\underset{G}{\Longrightarrow}} xv \text{ for some } u \in U \text{ and } v \in V\}. \blacksquare$$

Remark. (1) We consider here *left*-regular canonical systems, while [1], [5] and [11] consider *right*-regular canonical systems where a production (w, z) is applied in the fashion $w\alpha' \underset{G}{\Longrightarrow} z\alpha'$ (rather than $\alpha' w \underset{G}{\Longrightarrow} \alpha' z$ as in the above definition). As easily seen (and remarked already in [1]) this does not matter as far as the generative power of systems is concerned. (2) What we call the language of G is in [1] referred to as the *language produced by* G and in [11] as the *language generated by* G. Obviously the reasoning analogous to the one below (the proof of Corollary 2) goes also for what is referred in [1] and [11] as the *language accepted* by G. ■

Corollary 2. Let $G = (\Omega, \Phi, U, V, P)$ be a regular canonical system. If U, V are regular, then $L(G)$ is regular.

Proof. (sketch)

A 2faupd $\mathbf{A}_G$ accepting $L(G)$ is constructed as follows.

A word $x \in \Omega^+$ is placed on the input tape. $\mathbf{A}_G$ starts by generating an arbitrary word $u \in U$ on its push-down tape ($\mathbf{A}_G$ "remembers" U in its state structure).

From this moment on **A** iterates the following procedure. It chooses (non-deterministically) a production $\pi_1 = (w_1 , z_1)$ and checks whether w is a suffix of the current push-down tape word (it starts with $y_0 = u$). If not, then $\mathbf{A}_G$ gets "stuck"; if yes then it replaces the (erased) suffix w by the word z - let y_1 be the so obtained word on the push-down tape. Now starting with y_1 the above step (choosing a production π_2 and trying to apply it) is repeated yielding (if $\mathbf{A}_G$ have not got "stuck") y_2 on the push-down tape. In this fashion the sequence of words $y_0(= u), y_1 , y_2 , \cdots$ is produced on the push-down tape.

At any point after a y_i is obtained, $i \geq 0$, $\mathbf{A}_G$ may choose to switch to the "V-checking mode". That is, $\mathbf{A}_G$ goes down the push-down tape checking whether or not it contains a suffix from V (this is easily done because $\mathbf{A}_G$ "remembers" V in its state structure). If not $\mathbf{A}_G$ gets stuck; if yes (let v be the "found" suffix from V), then $\mathbf{A}_G$ (after erasing v "while checking") switches to the "input-checking mode". That is $\mathbf{A}_G$ checks whether or not the word on the push-down tape is identical to the word on the input tape (hence w). If not, then $\mathbf{A}_G$ gets stuck; if yes then $\mathbf{A}_G$ goes off the input tape to the left and accepts.

Clearly $x \in L(\mathbf{A}_G)$ if and only if $x \in L(G)$.

Now if $\Lambda \in L(G)$, then the above construction is modified in the obvious way so that $\Lambda \in L(\mathbf{A}_G)$. Consequently, for each $\alpha \in \Omega^*$, $\alpha \in L(\mathbf{A}_G)$ if and only if $\alpha \in L(G)$.

Hence, by Theorem 2, $L(G)$ is regular. ∎

Remark.

One may generalize (right−)regular canonical systems by allowing productions of the form $R_1 \to R_2$, where R_1, R_2 are regular languages. Such a production can be applied to a word α if α has a suffix in R_1. If a production $R_1 \to R_2$ can be applied, then one chooses a suffix in R_1 and replaces it by a word in R_2.

The language of such a generalized system is defined as in the case of (right−)regular systems.

It is easily seen that also these systems are easily simulated by 2faupd's (actually the simulation technique is very closed to the one from the proof outlined above). Thus also such generalized systems generate only regular languages providing their "start" and "end" sets are regular. (One may notice here that proving *directly* in the framework of regular canonical systems that so generalized regular canonical systems generate regular languages only is quite cumbersome). ■

5. OTHER TYPES OF WORKING TAPES.

In this section we consider other (than push-down) types of working tapes working in unison with a 2-way finite automaton.

Perhaps the most natural generalization of a 2 faupd model is to attach to a 2-way finite automaton a stack tape - that is a push-down tape on which it is allowed to go inside the push-down tape (without erasing top symbols) providing that such an entry is "read-only" (see, e.g., [6]).

Such a model is formalized as follows.

Definition. A 2faust $\mathbf{A} = (\Delta, \Sigma, Q, F, q_{in}, \delta)$ is called a *2-way finite automaton with a unison stack tape*, a 2faust for short, if the following holds.

(1) $Q = Q_P \bigcup Q_R$ with $Q_P \bigcap Q_R = \emptyset$, $q_{in} \in Q_P$ and $F \subseteq Q_P$; elements of Q_P are called *push-down states* and elements of Q_R are called *reading states.*

(2) Let (x, y, q) be a situation and (y', n, q') be an action such that $(y', n, q') \in \delta(x, y, q)$. Then

(2.1) *(Bouncing off the left end marker)*

If $x = \$$, then $q \in Q_R$ and $n = +1$.

(2.2) *(Switching form push-down to reading states)*

If $q \in Q_P$ and $q' \in Q_R$, then $y \neq B$, $y' = y$ and $n = 0$.

(2.3) *(Reading)*

If $q, q' \in Q_R$, then $y \neq B$ and $y' = y$.

(2.4) *(Switching back to push-down states)*

If $q \in Q_R$ and $q' \in Q_P$, then $y = B$, $y' = B$, $n = 0$ and moreover, also $(y, n, q') \in \delta(x', y, q)$ for all $x' \in \Delta \bigcup B$. ∎

Remark. We would like to make the following comments about the above definition.

(1) The states of Q are divided into push-down states (Q_P) and reading states

(Q_R). The states from Q_P are as the states of a 2faupd, the state from Q_R are reading only states. Each accepting computation starts in Q_P ($q_{in} \in Q_P$) and ends in Q_P($F \subseteq Q_P$).

(2.1) The only way not to get stuck on the left-end-marker \$ is to enter it in a reading state - then the only action possible is to bounce off to the right in a reading state. (The reader may see now a motivation behind our choice of conditions on reading \$ in the definition of a 2faupd.)

(2.2) Switching from a push-down state to a reading state can happen only if the top of the stack is different from B. Such a switch leaves the head in the same position and does not change the top symbol of the stack.

(2.3) The reading happens only inside the stack (i.e., not on the blank symbol) and it leaves the symbols of the stack unchanged.

(2.4) Switching back to a push-down state may happen only at the top of the stack when the top is blank. Such a switch leaves the head in the same position, the top symbol (B) unchanged and it does not depend on the current symbol on the input tape (i.e, **A** will switch from q to q' independently of the current input symbol.). ▪

The class of languages of all 2faust's is denoted by $\mathbf{L}(2FAUST)$.

We will demonstrate that 2faust's define regular languages. This will be done by a "reduction": we will show how for every 2faust one can construct a 2faupd defining the same language. The basic technical notion used in such a reduction is the notion of the leaving record (of a 2faust **A**).

Intuitively speaking, given a 2faust $\mathbf{A} = (\Delta, \Sigma, Q_P \cup Q_R, F, q_{in}, \delta)$ its *leaving record* $\Theta_{\mathbf{A}}$ provides for each pair ($w \in \Delta^* B^*$, $z \in \Sigma^*$) with $|w| = |z|$ the set of pairs from $Q_R \times Q_R$ such that:

if (q_1, q_2) is in $\Theta_{\mathbf{A}}(w, z)$, then there is a computation $\mu = \tau_0, \tau_1, \ldots, \tau_l$, $l \geq 1$, with $\tau_0 = (\$wB^\omega, \$zB^\omega, |w|, q_1)$, $\tau_l = (\$wB^\omega, \$zB^\omega, |w|+1, q_2)$ and

$pos(\tau_k) \leq |w|$ for all $0 \leq k < l$.

Thus starting in τ_0 on the last position (letter) of w and z in the reading state q_1 **A** may perform a computation in which it will leave for the first time w and z (to the right) in the reading state q_2.

This is formally defined as follows.

Definition. Let $\mathbf{A} = (\Delta, \Sigma, Q_p \cup Q_R, F, q_{in}, \delta)$ be a 2faust. The *leaving record (of* **A***)*, denoted $\Theta_{\mathbf{A}}$, is the function from $(\Delta^* B^*) \times (\Sigma^*)$ into $Q_R \times Q_R$ defined as follows. For $w \in \Delta^* B^*$ and $z \in \Sigma^*$:

(1) $\Theta_{\mathbf{A}}(\Lambda, \Lambda)$ is the set of all pairs $(q_1, q_2) \in Q_R \times Q_R$ such that $(\$, +1, q_2) \in \delta(\$, \$, q_1)$.

(2) If $w \in \Delta^+ B^*$, $z \in \Sigma^+$ and $|w| = |z|$, then $\Theta_{\mathbf{A}}(w,z)$ is the set of all pairs $(q_1, q_2) \in Q_R \times Q_R$ such that there exists a computation $\mu = \tau_0, \tau_1, \ldots, \tau_l$ such that $\tau_0 = (\$wB^\omega, \$zB^\omega, |w|, q_1)$, $pos(\tau_l) = |w| + 1$, $st(\tau_l) = q_2$, and for all $0 \leq k \leq l$, $pos(\tau_k) \leq |w|$.

(3) $\Theta_{\mathbf{A}}(w,z)$ is undefined in all others cases. ■

Perhaps the most important (for our purpose) property of the leaving record (of a 2faust **A**) is that it can be *updated* when going (to the right) from a given configuration τ to the next configuration τ': that is knowing $\Theta_{\mathbf{A}}((input(\tau), work(\tau))$ (with $|input(\tau)| = |work(\tau)|$ and the last letter of τ different from B) and a, b such that

$input(\tau') = (input(\tau))a$, $work(\tau') = (work(\tau))b$

one can compute $\Theta_{\mathbf{A}}(input(\tau'), work(\tau'))$. This is demonstrated in the next result (we provide only an outline of a procedure for such an update leaving formal details of the proof to the reader).

Lemma 3. Let $\mathbf{A} = (\Delta, \Sigma, Q_p \cup Q_R, F, q_{in}, \delta)$ be a 2faust. There exists a function $upd_{\mathbf{A}}$ from $Q_R \times Q_R \times (\Delta \cup B) \times \Sigma$ into $Q_R \times Q_R$ such that, for all $w \in \Delta^* B^*$ and all

$z \in \Sigma^*$ with $|w| = |z|$, $upd_{\mathbf{A}}(\Theta_{\mathbf{A}}(w,z),a,b) = \Theta_{\mathbf{A}}(wa,zb)$ for all $a \in \Delta \cup B$, $b \in \Sigma$.

Proof. (sketch)

This follows really directly from te definition of $\Theta_{\mathbf{A}}$.

Let $a \in \Delta \cup B$, $b \in \Sigma$ and consider $\Theta_{\mathbf{A}}(wa,zb)$ for an arbitrary $w \in \Delta^* B^*$ and $z \in \Sigma^*$ such that $|w| = |z|$.

(0) Let τ and ρ be configurations such that $\tau = (\$waB^\omega, \$zbB^\omega, |w|+1, q)$ and $\tau' = (\$waB^\omega, \$zbB^\omega, |w|+2, q')$, where $q, q' \in Q_R$.

If $\tau \vdash_{\mathbf{A}}^{+} \tau'$, then

(1) Either $\tau \vdash_{\mathbf{A}} \tau'$ and then (q, q') must be included in $\Theta_{\mathbf{A}}(wa,zb)$.

(2) Or $\tau \vdash_{\mathbf{A}} \tau'$ in more than one step.

Then $\tau \vdash_{\mathbf{A}} \tau_1 = (\$wB^\omega, \$zB^\omega, |w|, q_1')$ for some $q_1' \in Q_R$. For each such q_1' and for each q_1 such that $(q_1', q_1) \in \Theta_{\mathbf{A}}(w,z)$ we include in $\Theta_{\mathbf{A}}(wa,zb)$ pairs (q_1, q_1'') such that $(b, +1, q_1'') \in \delta(a,b,q_1)$.

(3) Then again, for each q_1 as above we consider all pairs of configurations $\tau_1 = (\$waB^\omega, \$zbB^\omega, |w|+1, q_1)$, $\tau_1' = (\$waB^\omega, \$zbB^\omega, |w|+2, q_1')$, for all $q_1' \in Q_R$, and for each such pair (if it was not already encountered before) we proceed as under (1) and (2) above adding pairs from $Q_R \times Q_R$ to $\Theta_{\mathbf{A}}(wa,wb)$. Clearly in a finite number of steps (not exceeding $\#(Q_R \times Q_R)$) no new pairs will be added to $\Theta_{\mathbf{A}}(wa,zb)$. If we perform this procedure for each pair $(q, q') \in Q_R \times Q_R$ we get$\Theta_{\mathbf{A}}(wa,zb)$.

Hence to construct $\Theta_{\mathbf{A}}(wa,zb)$ we need to know $\Theta_{\mathbf{A}}(w,z)$ and (a,b) only. Thus the lemma holds. ∎

We will demonstrate now that for each 2faust its language can be defined by 2faupd.

Lemma 4. $L(2FAUST) \subseteq L(2FAUPD)$.

Proof. (sketch)

We will show how for an arbitrary 2faust **A** one constructs a 2faupd **A'** such that $L(\mathbf{A}) = L(\mathbf{A}')$.

Let $\mathbf{A} = (\Delta, \Sigma, Q = Q_P \cup Q_R, F, q_{in}, \delta)$ be a 2faust.

Intuitively speaking the 2faupd $\mathbf{A}' = (\Delta', \Sigma', Q', F', q'_{in}, \delta')$ works as follows. Symbols of its working alphabet Σ' are triplets $[a, U, V]$ with $a \in \Sigma$ and $U, V \subseteq Q_R \times Q_R$. Then whenever **A** is in a situation τ with $input(\tau) = wa$, $work(\tau) = zb$, where $a \in \Delta \cup B$ and $b \in \Sigma$, and $st(\tau) \in Q_p$, **A'** will be in the situation τ' with $input(\tau') = wa$, $work(\tau') = z'[b, \Theta_{\mathbf{A}}(w,z), \Theta_{\mathbf{A}}(wz, zb)]$ for some z', $pos(\tau') = pos(\tau)$ and $st(\tau') = st(\tau)$.

If **A** goes from the situation τ as above to the right getting to a situation τ_1 with $input(\tau_1) = waa_1$ for some $a_1 \in \Delta \cup b$, $work(\tau_1) = zb'B$ and $st(\tau_1) \in Q_P$, then **A'** goes to the situation τ'_1 with $input(\tau'_1) = input(\tau_1)$, $work(\tau'_1) = z'[b', \Theta_{\mathbf{A}}(w,z), \Theta_{\mathbf{A}}(wa, zb')]B$, $pos(\tau'_1) = pos(\tau_1)$ and $st(\tau'_1) = \langle st(\tau_1), \Theta_{\mathbf{A}}(wa, zb')\rangle$ - to construct τ'_1, **A'** uses the update function $upd_{\mathbf{A}}$.

Hence we have two kinds of state: original state of **A** used only when the topmost symbol of the push-down tape in **A'** is not blank and "annotated" states of the form $\langle q, U\rangle$, where $q \in Q_P$ and $U \subseteq Q_R \times Q_R$ used only when the top of the push-down tape in **A'** is blank - in this situation q is the state of the corresponding situation in **A** and U is the leaving record (in **A**) of the portion of the tapes immediately to the left of the position of the heads.

Formally the 2faupd $\mathbf{A}' = (\Delta', \Sigma', Q', F', q_{in}, \delta)$ is defined as follows.

(1) $\Delta' = \Delta$,

(2) $\Sigma' = \{[a, U, V] : a \in \Sigma \text{ and } U, V \subseteq Q_R \times Q_R\}$,

(3) $Q' = Q \cup \{<q, U> : q \in Q \text{ and } U \subseteq Q_R \times Q_R\}$,

(4) $q_{in}' = q_{in}$, and

(5) the transition function δ' is defined as follows:

(5.1) for all situations (x,B,q) in G such that $q \in Q_p$,

$([y', U, upd_{\mathbf{A}}(U,x,y')] , +1, <q', udp_{\mathbf{A}}(U,x,y')>) \in \delta'(x, B<q, U>)$

for all $U \subseteq Q_R \times Q_R$ whenever $(y' , +1 , q') \in \delta(x, B, q)$,

and $(B, -1, q') \in \delta'(x, B, <q, U>)$ for all $U \subseteq Q_R \times Q_R$

whenever $(B, -1, q') \in \delta(x, B, q)$,

(5.2) for all situations (x, y, q) in G such that $y \neq B$ and $q \in Q_p$,

$([y', U, udp_{\mathbf{A}}(U,x,y')], +1, (q', upd_{\mathbf{A}}(U,x,y')) \in \delta'(x,[y,U,V],q)$

for all $U,V \subseteq Q_R \times Q_R$ whenever $(y', +1, q') \in \delta(x,y,q)$,

and $(B, -1, q') \in \delta(x,[y,U,V],q)$ for all $U,V \subseteq Q_R \times Q_R$

whenever $(B, -1, q') \in \delta(x,y,q)$,

(5.3) for all situations (x,B,q_{in}) in G,

$([y', \Theta_{\mathbf{A}}(\Lambda,\Lambda), upd_{\mathbf{A}}(\Theta_{\mathbf{A}}(\Lambda,\Lambda),x,y')], +1, <q', upd_{\mathbf{A}}(\Theta_{\mathbf{A}}(\Lambda,\Lambda),x,y'>) \in \delta'(x,B,q_{in})$,

whenever $(y', +1, q') \in \delta(x,B,q_{in})$,

and $(B, -1, q') \in \delta(x,B,q')$.

whenever $(B, -1, q') \in \delta(x,B,q_{in})$.

(5.4) for all $q, q' \in Q_P$, $x \in \Delta \cup B$, $y \in \Sigma$ and $U, V \subseteq Q_R \times Q_R$

$([y,U,V],+1,<q',V>) \in \delta'(x,[y,U,V],q)$

whenever there exist $s,s' \in Q_R$ such that

$(y,0,s) \in \delta(x,y,q)$, $(s, s') \in V$ and $(B,0,q') \in \delta(t,B,s')$

for all $t \in \Delta \cup B$.

It is rather easy but tedious to show that $L(\mathbf{A}) = L(\mathbf{A}')$.

Hence the result holds. ∎

Theorem 3. $\mathbf{L}(FAUST) = \mathbf{L}(REG)$.

Proof.

The theorem follows from Lemma 4 and from the obvious observation that $\mathbf{L}(REG) \subseteq \mathbf{L}(FAUST)$. ▪

One could extend the faust model by allowing the "reading without changes" also forward. That is, after switching to the "reading mode" an automaton may decide to go forward (i.e. to the right) on the input tape in order to "check some information" - then it would return to the top of the push-down tape and either switch back to a push-down state or again in a reading state it could either go to "read backward" (i.e., to the left of the top of the push-down tape) or "read forward" (i.e., to the right of the top of the push-down tape).

It is easily seen that this extended model may be simulated by the faust model and so such automata recognize regular languages only.

We end this section by considering briefly the case of the 2faut model when the working tape is the linearly bounded tape (in a classic sense).

Definition. A 2faut $\mathbf{A} = (\Delta,\Sigma,Q,F,q_{in},\delta)$ is called a *2-way finite automaton with a unison linearly bounded tape*, a 2faulb for short, if for each situation (x,y,q) and each action (y',n,q'), $x = B$ implies $n = -1$. ▪

The class of languages of all faulb's is denoted by $\mathbf{L}(FAULB)$.

The following result is obvious.

Theorem 4. $\mathbf{L}(FAULB) = \mathbf{L}(CS)$. ▪

DISCUSSION.

In this paper we have extended the classical 2-way finite automaton model by attaching to it an auxiliary storage tape working in unison with the input tape. We have demonstrated that so extended model does not gain in the accepting power - still only regular languages are accepted. We have also demonstrated some applications of our results (Section 4): e.g., we believe that (once our result on faupd automata is given) our proof of the extended Büchi theorem is very simple (if not trivial) - certainly, the proofs existing in the literature are rather involved.

We would like to conclude this paper by relating our proof techniques to those from [9] and [13].

Clearly our proof techniques are based on these from [9] and [13]. In both [9] and [13] one makes a very basic use of "*crossing relations*", i.e. sets of pairs of states corresponding to crossing - *forwards* and *backwards* - of a cut between two positions on a tape (moreover in [9] one uses "*crossing sequences*" i.e., the histories of crossing a given cut). The basic observation (trick) of both [9] and [13] is observing that crossing relations are "time-independent", i.e., once one may cross a cut (forwards or backwards) in a way corresponding to a pair of states (q, q') (q - before the crossing and q' after the crossing) the crossing of this type may always be repeated (provided we are at "the same place" in the state q). Technically this allows one to classify prefixes of all accepted words into a finite number of types (classes) and then use the Nerode theorem, see e.g., [9], to prove that the language accepted by a 2-way finite automaton is regular.

The time-independence of crossing relations does not hold in models we consider in this paper. More specifically:

(1) the *forward-backward crossing relations* (i.e., crossing a cut forward *into* a

state q and returning backwards *into* a state q') is time independent (it may always be repeated: whenever in the future we cross forward the same cut into the state q we can always go backward into state q'), (2) however, the *backward-forward crossing relations* (i.e., crossing a cut backward into a state q and returning forward into a state q') are not any more state independent (whether in the future after crossing backward the same cut into the state q we can go forward into the state q' depends on the currently scanned symbol of the auxiliary storage - and this could have been changed in the meantime!)

So, roughly speaking, the main observation (trick) behind our proofs is that the prefixes (up to the position of the common head) can be characterized by *finite sets of* forward-backward crossing relations and this "characteristic sets" can be updated while we move to the right. Hence everything can be remembered in (a finite number of) states and there is no need for going backward (and hence no need for the auxiliary storage!)

ACKNOWLEDGEMENTS.

The authors gratefully acknowledge the support of NSF grant MCS 83-05245.

REFERENCES.

[1] Büchi, J.R., "Regular canonical systems", *Arch. Math. Logik und Grundlagenforsch.*, v. 6, pp. 91-111, 1964.

[2] Engelfriet, J., Schmidt, E.M. and van Leeuwen, J., "Stack machines and classes of nonnested macro languages", *Journal of the Assoc. for Comp. Mach.*, v.27, pp. 96-117, 1980.

[3] Engelfriet, J., Rozenberg, G. and Slutzki, G., "Tree transducers, L systems and two-way machines", *Journal of Computer and Systems Science*, v. 20, pp. 150-202, 1980.

[4] Greibach, S., "Checking automata and one-way stack languages", *Journal of Computer and Systems Science*, v.3, pp. 196-217, 1969.

[5] Greibach, S., "A note on push-down store automata and regular systems", *Proc. Amer, Math. Soc.*, v. 18, pp. 263-268, 1967.

[6] Harrison, M.A., *Introduction to formal language theory*, Addison-Wesley, Reading, Mass. 1978.

[7] Kratko, M.I., "Formal Post systems and finite automata (in Russian)", *Problemy Kibernet.*, v. 17, pp. 45-65, 1966.

[8] Post, E.L., "Formal reductions of the general combinatorial decision problem", *Amer. J. Math.*, v. 65, pp. 197-215.

[9] Rabin, M.O. and Scott, D., "Finite automata and their decision problems", *IBM Journ. of Res. and Develop.*, v. 3, pp. 114-125, 1959.

[10] Rozenberg, G., "On coordinated selective substitutions: Towards a unified theory of grammars and machines", *Theoretical Computer Science, to appear.*

[11] Salomaa, A., *Theory of automata*, Pregamon Press, Oxford, 1969.

[12] Salomaa, A., *Formal languages*, Academic Press, London-New York, 1974.

[13] Shepherdson, J.C., "The reduction of two-way automata to one-way automata", *IBM Journ. of Res. and Develop.*, v. 3, pp. 198-200, 1959.

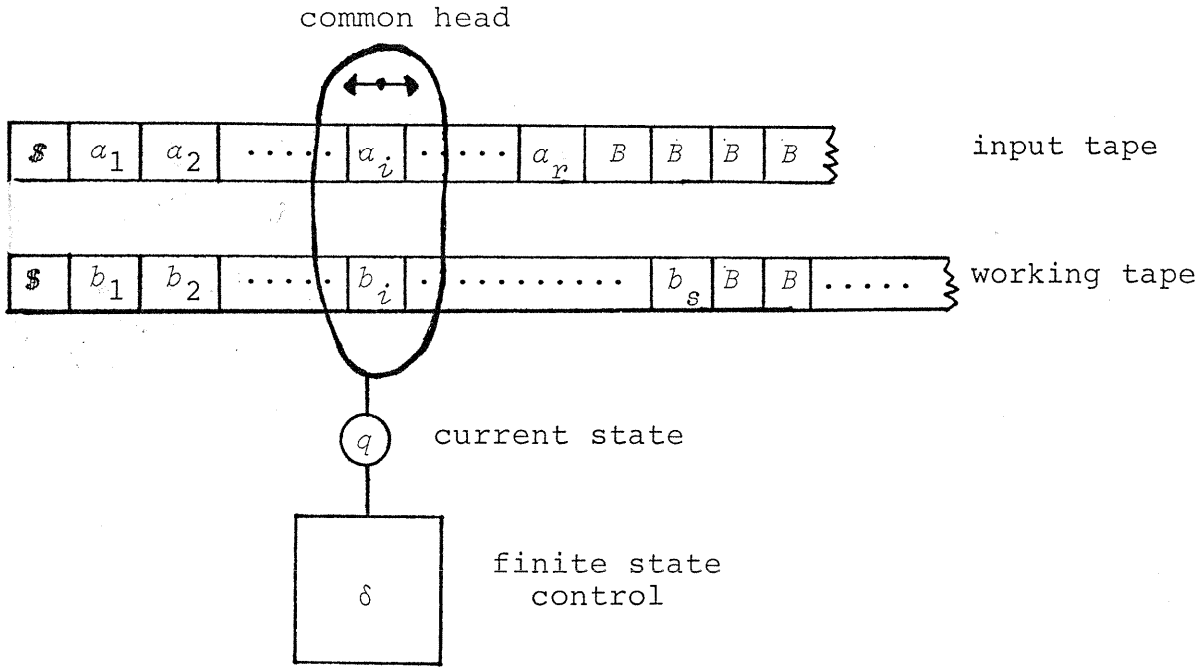
common head
$
a_1
a_2
a_i
a_r
B
input tape
$
b_1
b_2
b_i
b_s
B
working tape
q
current state
δ
finite state control

Figure 1

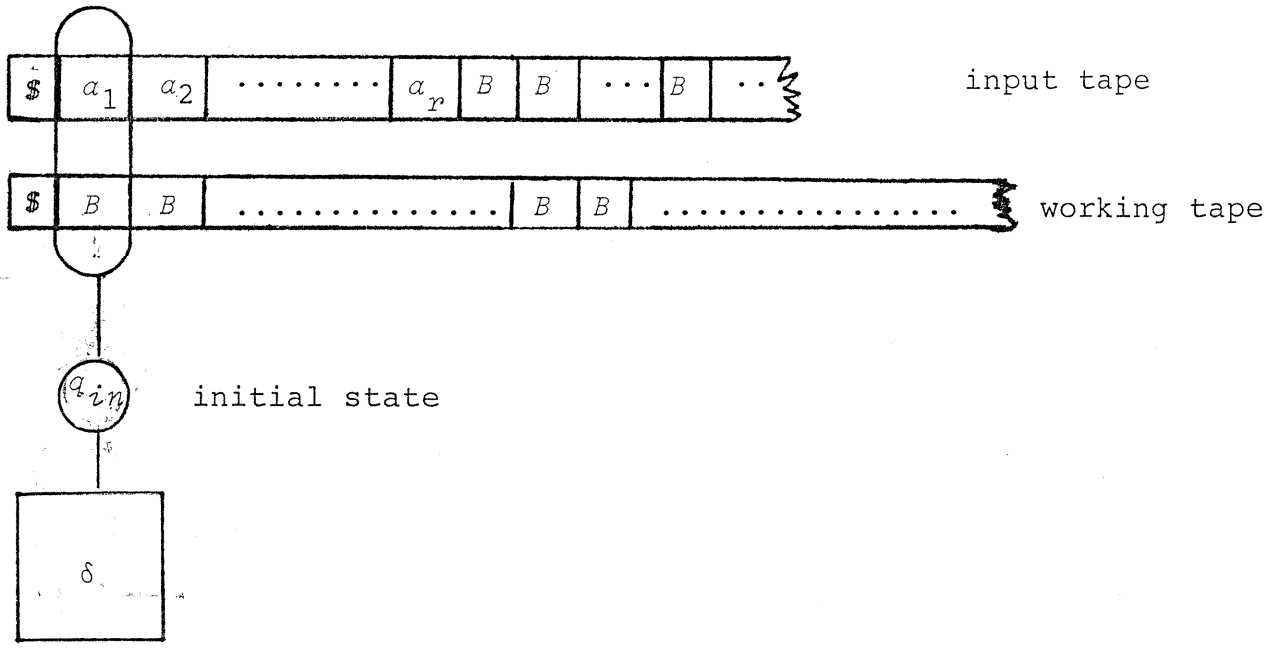

Figure 2

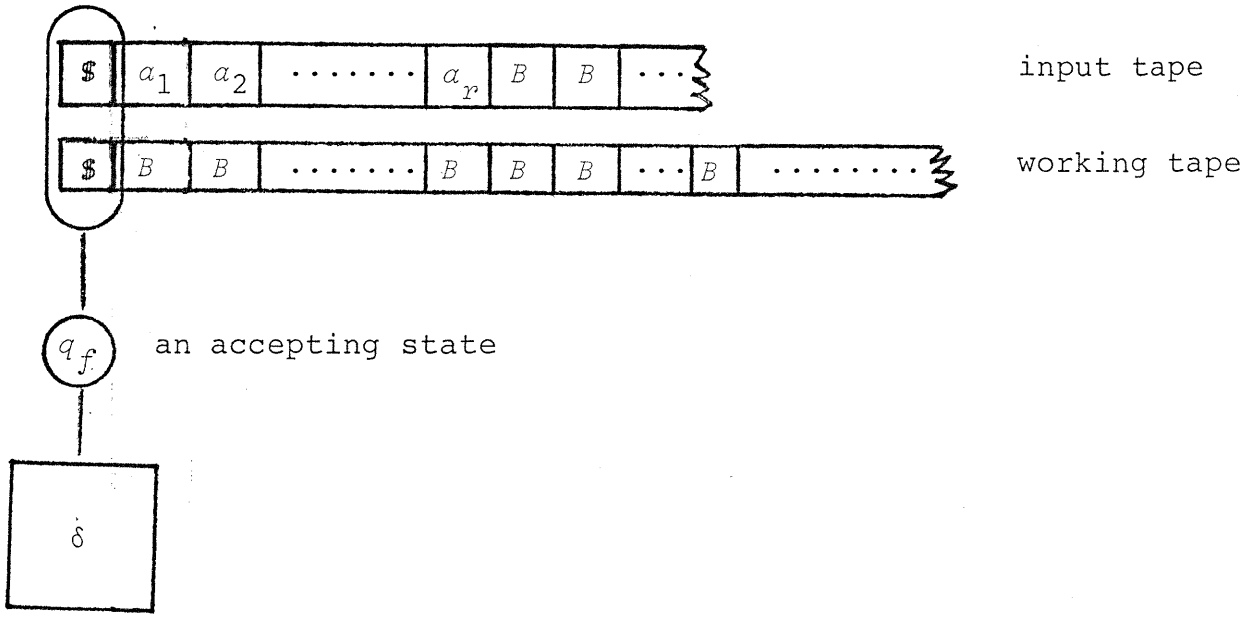

Figure 3